Efficient Foreground Segmentation Using Adaptive Clusters

Michael Seaholm
University of Wisconsin, Madison
seaholm@wisc.edu

Emma Turetsky
University of Wisconsin, Madison
eturetsky@wisc.edu

Abstract

For this project, we implemented an algorithm originally detailed by Bulter et al. *[1] in order to produce a system which can perform background-foreground segmentation with reasonable accuracy in close to real-time. The clustering algorithm described previously acts to provide historical information on a per-pixel basis, allowing for an adaptive background model. By measuring the difference between incoming video frames and this model as outlined by the algorithm, our system is able to classify pixels in the frame as either background or foreground. Besides this, post-processing was applied to the image mask calculated by our system to account for false positives and negatives, with varying degrees of success. Our results show a clearly separated foreground image that is largely free of defects, with a few notable exceptions. Although the nature of the system is robust in many respects to changes in the scene, avenues for improvement are myriad and could be the subject of further investigation.*

1. Introduction

The problem of automatically dividing an image up into background and foreground regions as applied to static images is fairly straightforward and has been covered extensively in the relevant literature, but when dealing with a live video sequence the task becomes more complicated. This is due primarily to two factors: the presence of motion in the background and the necessity of having these systems run accurately in real time.

Although many such systems use a stereo camera setup in order to capture depth information for the purposes of background-foreground segmentation, others, such as the one created by Butler *et al*. [1], are designed to use only one camera. Such systems typically use aspects of the video, such as color and motion, to determine the relative positions of each layer. For this project we expanded on the work of Butler and others to produce a system that segments live video into background and foreground layers quickly and with reasonable rates of accuracy using only one camera.

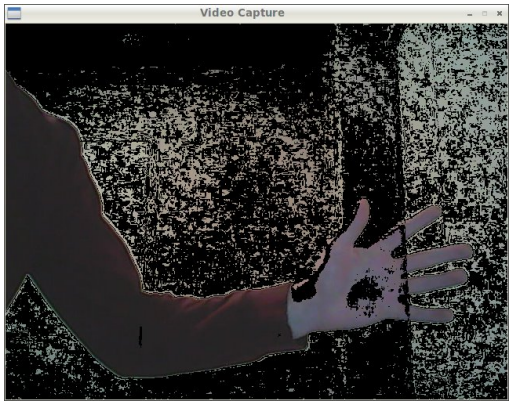

Figure 1. A frame of segmentation using the naive approach.

As this algorithm classifies each pixel independentally, it allows for parallelization during the classification process.

2. Naive Apporach

As an initial exercise in working with these resources, we started with a nave implementation of background-foreground segmentation. By taking a known image of the background, we can compare this model with frames of the incoming video and classify those pixels which differ significantly (i.e., exceeding a user-defined threshold) as the foreground. This approach suffers from a number of issues, chiefly that it is very sensitive to changes in luminance in the scene and to any motion which may occur in the scene. This implementation was used as a stepping stone to attain the actual purpose of our project and serves as an object of basic comparison with our results (see Figure 1).

3. Algorithm

We implemented the algorith described in [1]. The core idea behind the algorithm that our system has implemented is to model each pixel in the frame with a group of clusters, each member of which contains a weight, w and centroids Y, U, and V corresponding to components of the YUV color space. As incoming pixels are matched to a cluster

within its group, a history of the pixel's likelihood of being in the foreground is effectively being accrued and adapted based on past and future pixels. This approach proves to be robust to both changes in luminance and, in the long term, to background motion.

When the system is first run, it initializes all the cluster groups for each pixel in the canonical frame. As video frames are captured from the camera, each incoming pixel, p, and its corresponding cluster group, CG, is processed as follows:

First, the algorithm looks at each of the clusters in the group, sorted from highest to lowest weight, and finds the best match for the incoming pixel. The quality of the match is determined by looking at the difference between the YUV centroids in a given cluster and the corresponding values for the pixel. If each of these distances are below a specified threshold, then the cluster is considered a match. It should be noted that in our implementation, a separate threshold for each of the YUV components is established to allow greater control over matches made by the algorithm.

If a match is found, then the centroids of the matching cluster are adapted toward the incoming pixel by an amount specified by the learning rate, L. Similarly, the weights of all clusters in the group are adapted in accordance with this learning rate. These adaptations are given by the following equations. For the matching cluster group:

$$CG.Y = CG.Y + \frac{p.Y - CG.Y}{L}$$
$$CG.U = CG.U + \frac{p.U - CG.U}{L}$$
$$CG.V = CG.V + \frac{p.V - CG.V}{L}$$
$$CG.w = CG.w + \frac{1 - CG.w}{L}$$

For the non-matching clusters:

$$CG.w = CG.w - \frac{CG.w}{L}$$

If no match could be found, then the lowest weighted cluster is replaced by a new cluster whose centroids are set to the YUV values of the pixel and with an initial weight of 0.01. This effectively allows lower-weight clusters to expire as new and different pixel information comes in from the scene.

After the matching and adaptation stages have occurred, the weights of the clusters are normalized to ensure consistency between cluster groups and to facilitate the later classification stage. It should be noted that normalization need only occur when cluster replacement occurs during the matching stage, since the sum of weights across clusters remains 1 even after adaptation. Accounting for this helps to increase our implementation's overall efficiency.

During the classification stage, the clusters are sorted from highest to lowest weight and the algorithm computes the sum of weights for all clusters that are heavier than the one that matched the incoming pixel. Given that the weights have been normalized previously, this sum acts as a proxy for the probability that the given pixel is in the foreground. As such, by establishing a probability threshold, we can create a binary mask which reflects the classifications for each of these pixels. Our system takes this mask, applies it to the input frame, and outputs the resultant image, which contains the foreground region from the original and a blacked-out background region.

3.1. Post Processing

After the creation of the mask, our system performs additional post-processing steps to ensure the accuracy of the computed foreground region. As the problem can be viewed as a matter of binary classification, the process described above suffers primarily from two types of errors: background pixels incorrectly classified as foreground pixels and vice-versa. For the most part, the issue of false positives comes about due to noise in the capture device which causes occasional pixels to rise above one or more of the YUV thresholds across clusters in the group, which is interpreted as being in the foreground as a result. To remedy this, our system performs a morphological open operation on the foreground mask, which has the effect of eliminating small pockets of false positives which may occur in the background.

Although this approach produces a much smoother foreground mask, it is not without its limitations. Regions of false positives which are larger than several pixels in size and are not isolated from other foreground regions are reduced, but not eliminated, by this operation. This is most evident in the presence of shadows from foreground objects in the scene, as they produce a larger change in luminance where they are cast than is typically accounted for by regular fluctuations in lighting. Empirically, however, we've found that careful adjustment of the luminance threshold reduces the classification of shadows as foreground sufficiently for normal purposes.

False negatives, meanwhile, occur primarily in regions in which there is almost no difference between the foreground pixels and the adapted background model across all three YUV components. In most environments in which such systems would be used, matches tend to occur along the extended edges of foreground objects. We chose to address the problem of false negatives by performing a morphological open operation on the bitwise inversion of the foreground mask, which has the property of smoothing the affected regions and resolving the bulk of the issue.

Sometimes, however, the result of the algorithm contains connected background regions, or holes, within the fore-

Figure 2. Two frames showing correct foreground/background segmentation of the images.

ground mask due to due to across-the-board similarity between background and foreground pixels. While the Butler paper indicates the use of a connected components-based algorithm to correct for such instances, our experimental results indicated few circumstances in which such holes would occur which would not be corrected over time. As such, our implementation differs in that regard.

4. Evaluation

4.1. Implementation

We implemented our system in C++ using the OpenCV computer vision libraries and used a Logitech C110 webcam for video capture. The performance evaluation is on a maching running 64-bit Redhat 6 Linux with an Intel Core 2 Quad Processor running at 2.66 Ghz with 8GB of RAM. Our code is located at http://pages.cs.wisc.edu/~seaholm/CS766/project.html

4.2. Results

The output of our system is represented by the series of still frames in Figure 2

Each frame is 640 x 480 pixel resolution taken from the video output stream that our system produces. The background portion of these frames is blacked out as per our implementation to accentuate the foreground regions and to facilitate finding false positives and negatives as they occur. It should be noted that these frames were taken after first letting the algorithm run with no foreground objects in the scene so as to provide an initial stable background model. Although the system is capable of adapting to changes in the background regardless of initial state, this was done to avoid having to wait for the weights to shift appropriately due to adaptation.

Originally, the system ran slowly owing to the computationally-intensive nature of the algorithm; preliminary results indicated that on average processing for each frame took about 340 ms, which translates to about 3 frames per second. We strove to improve the running time by simplifying certain aspects of the algorithm. As mentioned in the original paper by Butler *et al*., the normalization stage does not need to be performed every iteration as we had done previously; since the adaptation function for the cluster weights did not change the sum of weights in the cluster group, normalization need only occur when no cluster match could be found and a new cluster with a preset weight was created. As well, by writing certain simple but frequently-referenced operations as preprocessor macros and setting optimization flags for the compiler and implementing parallelization, we saw a significant increase in performance. The present iteration of the system takes about 45 ms to process a frame, or about 22 frames per second, which is roughly real-time.

4.3. Parallel Optimization

We also implemented a version of the algorithm that split the workload of the foreground classification into four separate threads. This gave us a speedup of 25% during that portion and one of 18% overall. Further experiments with 8 and 16 threads showed no appreciable difference.

4.4. Discussion

The results that we have seen from our system illustrate both the accuracy and robustness of the underlying cluster-based algorithm. The foreground is clearly separated from the background and appears to be accurate to a high degree; speckling phenomena from false positives in particular have been smoothed out surprisingly well by the morphological open, producing a background region that is largely contiguous. The same operation as applied to false negatives is successful, although not to the same degree; small holes do still occur within the foreground image and along sharp edges that may be mistaken as the background due to reflection and other visual similarities.

Perhaps the most important aspect of the algorithm is its ability to adapt its model of the background over time, as is reflected in Figure 3. Objects in the scene which cause very little in the way of pixel change (e.g., motionless or very

Figure 3. A frame showing the model adapting and classifying a still object as the background.

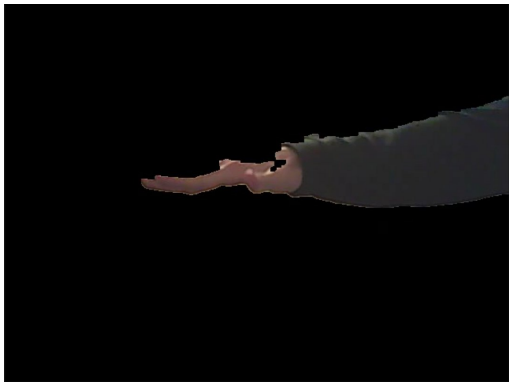

Figure 4. This figure shows errors in determining the edges of the foreground hand.

slow moving objects) eventually become incorporated in the background model, the rapidity of which being contingent upon the learning rate specified by the user. This allows for more accurate segmentation of the foreground from the background, especially in the long term.

Of course, our implementation is not without its imperfections. As mentioned previously, certain extreme luminance differences, such as shadows, tend to be interpreted as foreground regions unless the luminance threshold is carefully tuned. Even then, small shadows may persist unless lighting parameters within the environment are strictly controlled, which is an undesirable constraint. Finding a way to identify shadows in the scene and classifying them as being part of the background is chief among the features which should be implemented in future iterations of the implementation.

Besides the issue of shadows being classified as false positives, the issue of false negatives in the form of holes in the foreground region and choppiness around certain edges of the foreground objects, such as in Figure 4, still persist. For the former issue, the connected-components approach mentioned previously would likely handle it better than we do at present. As for the latter, it may be that a later implementation could check the edges of foreground objects and smooth them to reduce the choppy effect, but since the issue comes about due to a similarity between the background and the foreground, it is difficult to say whether or not the edges in those regions could be reliably discerned.

5. Conclusion

The foreground segmentation system that we implemented, based on the algorithm developed by Butler *et al*., uses an adaptive clustering technique to build a historical record of the pixels in a given scene, which provides a background model which slowly changes over time to account for changes in the environment. This proves to be robust to changes in luminance and motion and, for the most part, properly captures the foreground region from the background. As we have previously detailed, the system is not error-proof, as false positives and false negatives are bound to occur owing to their being inherent to the problem of background-foreground segmentation. We have taken steps to minimize and, in some cases, eliminate instances of these errors, and toward this end different additional approaches could be applied to produce even more accurate results. With regards to system performance, the move to parallelization provided a significant boost in terms of the frame rate, and so further investigations along these lines might also prove beneficial in future research. Given our results, it is clear that the system which we have produced lends itself to modification to provide for these possible improvements.

References

[1] D. E. Butler, V. M. Bove, Jr., and S. Sridharan. Real-time adaptive foreground/background segmentation. *EURASIP J. Appl. Signal Process.*, 2005:2292–2304, Jan. 2005. 1